Практическая работа №2. Отчёт

Команда №1
Состав: Азимжанова Инаара (группа 312), Банников Дмитрий (группа 311), Ройтман Андрей (группа 311).

Отчёт содержит:
- описание теории, используемой при написании программы, в разделе *Постановка задачи*;
- используемый для вычислений в программе симплекс-метод, описывается отдельно в одноимённом разделе в общем виде;
- описание написанного кода для выполнения задачи в разделе *Исполнение*.

Постановка задачи

Пусть функция $F(x, y)$ определена на декартовом произведении $X \times Y$, где X, Y — множества произвольной природы.

Определение. Пара $(x^0, y^0) \in X \times Y$ называется *седловой* точкой функции $F(x, y)$ на $X \times Y$, если

$$F(x, y^0) \le F(x^0, y^0) \le F(x^0, y) \text{ для любых } x \in X, y \in Y$$

или, эквивалентно,

$$max\, F(x, y^0) = F(x^0, y^0) = min\, F(x^0, y), \; x \in X, y \in Y$$

Опишем антагонистическую игру. В ней принимают участие два игрока 1 и 2 (первый и второй). Игрок 1 выбирает стратегию x из множества стратегий X, игрок 2 выбирает стратегию y из множества стратегий Y. Нормальная форма игры подразумевает, что каждый игрок выбирает свою стратегию независимо, не зная выбора партнера. Задана функция выигрыша $F(x, y)$ первого игрока, определенная на $X \times Y$. Выигрыш $F(x, y)$ первого игрока является проигрышем для второго. Цель первого игрока состоит в увеличении своего выигрыша $F(x, y)$, а цель второго — в уменьшении $F(x, y)$.

Таким образом, антагонистическая игра задается совокупностью $\Gamma = (X, Y, F(x, y))$.

Определение. Говорят, что антагонистическая игра Γ имеет решение, если функция $F(x, y)$ имеет на $X \times Y$ седловую точку. Пусть (x^0, y^0) — седловая точка функции $F(x, y)$. Тогда тройка $(x^0, y^0, v = F(x^0, y^0))$ называется *решением игры*, (x^0, y^0) — *оптимальными стратегиями игроков,* а v — *значением игры.*

Определение. Антагонистическая игра Γ называется *матричной*, если множества стратегий игроков конечны: $X = \{1, ..., m\}$, $Y = \{1, ..., n\}$. При этом принято обозначать стратегию первого игрока через i, стратегию второго через j, а выигрыш первого $F(i, j)$ через a_{ij}. Матрица $A = (a_{ij})_{m\times n}$ называется *матрицей игры*. Первый игрок выбирает в ней номер строки i, а второй — номер столбца j. В обозначениях матричной игры (i^0, j^0) — *седловая* точка матрицы A, если

$$a_{ij^0} \le a_{i^0j^0} \le a_{i^0j}, \ i = 1, ..., m, \ j = 1, ..., n.$$

Рассмотрим игру Γ с точки зрения первого игрока. Пусть он выбрал стратегию x. Ясно, что его выигрыш будет не меньше, чем $inf\, F(x, y)$ при $y \in Y$. Величину $inf\, F(x, y)$ при $y \in Y$ назовем *гарантированным результатом* (выигрышем) для первого игрока, если он использует стратегию x. Наилучший гарантированный результат для первого игрока задается формулой

$$\overline{v} = sup\, inf\, F(x, y), \ (sup : x \in X, \ inf : y \in Y)$$

называется *нижним значением игры.*

Определение*.* Стратегия x^0 первого игрока называется *максиминной*, если $inf\, F(x^0, y) = \overline{v}, \ y \in Y$.

Рассмотрим игру Γ с точки зрения второго игрока. Если он выбрал стратегию y, то для него естественно считать гарантированным результатом величину $sup\, F(x, y)$ при $x \in X$. Проигрыш второго игрока будет не больше, чем эта величина. Наилучший гарантированный результат для второго игрока задается формулой $\widehat{v} = inf\, sup\, F(x, y) \ (sup : x \in X, \ inf : y \in Y)$ и называется *верхним значением игры.*

Определение*.* Стратегия y^0 второго игрока называется *минимаксной*, если $sup\, F(x, y^0) = \widehat{v}, \ x \in X$.

Сведем решение матричной игры к паре двойственных задач линейного программирования.

Значение матричной игры представимо в виде:

$$v = max\, min\, A(p, j) = max\, min \sum_{i=1}^{m} p_i a_{ij}, \ p \in P, \ 1 \le j \le n.$$

Введем вспомогательную переменную *u* и запишем задачу нахождения максимума как задачу ЛП

$$v = max\, u,\ (u,\ p) \in B,$$

где

$$B = \{(u,\ p) \mid \sum_{i=1}^{m} p_i a_{ij} \geq u,\ j = 1, ..., n,\ \sum_{i=1}^{m} p_i = 1,\ p_i \geq 0,\ i = 1, ..., m\}.$$

Действительно, при фиксированном $p \in P$ максимальное значение u при ограничениях $(u,\ p)$ равно

$$min\, A(p,\ j),\ 1 \leq j \leq n.$$

Поскольку $v > 0$, можно считать, что u принимает положительные значения. Сделаем замену переменных $z_i = p_i/u,\ z = (z_1, ..., z_m)$. Тогда, учитывая ограничения $(u,\ p) \in B$, получим

$$\sum_{i=1}^{m} z_i = 1/u,\ \sum_{i=1}^{m} a_{ij} z_i \geq 1,\ j = 1, ..., n,\ z_i \geq 0,\ i = 1, ..., m.$$

Отсюда

$$v = max\, u = 1/\sum_{i=1}^{m} z_i^0,\ (u,\ p) \in B,$$

где z^0 — оптимальное решение задачи ЛП

$$\sum_{i=1}^{m} z_i \to min$$

(1)

$$\sum_{i=1}^{m} a_{ij}\ z_i \geq 1,\ j = 1, ...n,\ z_i \geq 0,\ i = 1, ..., m.$$

По z^0 находим значение игры и оптимальную смешанную стратегию первого игрока:

$$v = 1/\sum_{i=1}^{m} z_i^0,\ p^0 = vz^0.$$

Аналогично можно получить, что

$$v = min\, max\, A(i,\ q) = 1/\sum_{j=1}^{n} w_j^0,\ q \in Q,\ 1 \leq i \leq m,$$

где w^0 — оптимальное решение задачи ЛП.

$$\sum_{j=1}^{n} w_j \rightarrow max$$

$$\sum_{j=1}^{n} a_{ij} w_j \le 1,\ i = 1, ..., m,\ w_j \ge 0,\ j = 1, ..., n. \tag{2}$$

Здесь $q^0 = vw^0$ — оптимальная смешанная стратегия второго игрока.

Задачи (1) и (2) двойственны одна по отношению к другой.

Симплекс-метод

Для решения задач линейного программирования использовался симплекс метод. Алгоритм симплекс метода основан на переборе вершин канонического полиэдра(соответствующего нашей задаче) и поиск среди них оптимальной вершины. Решение общей распределительной задачи выполняется в два этапа:

1. Находят любое решение (как правило, неоптимальное), удовлетворяющее ограничениям, или убеждаются, что решения не существует. Этот этап называется определением опорного плана (базиса).
2. Производится последовательное улучшение данного плана до получения оптимального. В некоторых задачах опорный план определяется легко, в противном случае используют специальные методы получения опорного плана.

<u>Описание алгоритма Симплекс метода:</u>

1. Выражение целевой функции через небазисные переменные: f (x) = 5x1 +6x2. Записать целевую функцию в форме Таккера: f (x) = 0−(−5x1 −6x2).
2. Проверка базисного решения на оптимальность.
3. Проверка на наличие решения.
4. Выбор из небазисных переменных той, которая способна при введении её в базис увеличить значение целевой функции
5. Определение, какая из базисных переменных должна быть выведена из базиса.
6. Выражение вводимой в базис переменной через выводимую и другие небазисные переменные.
7. Выражение остальных базисных переменных и целевой функции через новые небазисные переменные.
8. Повторение операций пунктов (2) - (7) до тех пор пока не будет найдено оптимальное решение (или не будет показано, что решения нет).

Для реализации симплекс метода была использована библиотечная функция linprog(), которая использует усовершенствованный симплекс метод (подбор стартовой вершины и опорного базиса)

Исполнение

nash_equilibrium (a)

Используемые библиотеки. *scipy.optimize* (импортируется одна функция *linprog()*), *fractions* (импортируется один класс *Fraction*).

Ввод. Матрица *a*, являющаяся матричной игрой.

Задача. Производит обработку вводимой матричной игры, то есть решает её методом разбиения на две двойственные задачи.

Возвращаемые значения. Массив, содержащий три элемента:

1. объект класса Fraction, представляющий гарантированный выигрыш от игры,
2. массив объектов типа *Fraction,* представляющий вектор оптимальных смешанных стратегий первого игрока,
3. массив объектов типа *Fraction*, представляющий вектор оптимальных смешанных стратегий второго игрока.

Работа функции. В самом начале вычисляется количество строк и столбцов вводной матрицы для использования в дальнейших вычислениях. Затем создаются следующие объекты для двух двойственных задач: *cz*, *bz*, *cw*, *bw*, *az*, *aw*. Тройка *cz*, *bz*, *az* используется для решения задачи (1). Соответственно, тройка *cw*, *bw*, *aw* используются для решения задачи (2), двойственной первой. В обоих случаях:

- одномерный массив с названием вида *c** представляет вектор, скалярное произведение неизвестных с которым надо минимизировать (в случае c * = *z*) или максимизировать (в случае c * = *w*),
- одномерный массив с названием вида *b** представляет вектор правых частей неравенств задачи линейного программирования,
- двухмерный массив с названием вида *a** представляет матрицу, задающую правые части неравенств задачи линейного программирования.

Затем к обоим тройкам объектов применяется функция *scipy.optimize.linprog(c*, a*, b*)* (далее *linprog()*) для решения соответствующих им задач линейного программирования симплекс-методом. Стоит отметить, что условие неотрицательности решения при неуказанных других условиях на решение предусмотрено функцией *linprog()* по умолчанию и что из-за ограничений функции понадобилось создавать объекты *c*, a*, b** в виде, отличном от теоретической формулировки задач линейного программирования, то есть:

- матрица *az* была транспонирована и умножена на *-1* (для получения матрицы *az* вводная матрица *a* сперва транспонируется функцией *matrix_t(a)*, а после её элементы умножаются на *-1* с функцией *matrix_negative(a)*), в вектор *bz* был составлен из *-1*, а не *1*, чтобы поменять знак неравенств в теоретической формулировке с >= на <=, поскольку функция *linprog()* обрабатывает систему неравенств только со знаком <=,

- вектор *cw* был составлен из *-1*, чтобы поменять цель задачи с максимизации на минимизацию, поскольку функция *linprog* ищет решение на минимизацию исходного выражения.

Результат работы функции *linprog()* записывается в переменных *z_res* и *w_res* (для соответствующих вызовов функций) типа *scipy.optimize.OptimizeResult*, и в случае, если для обеих задач симплекс-методом было найдено оптимальное решение (то есть булевая переменная *success* в классе *scipy.optimize.OptimizeResult* имеет значение *true*), то в массивы *z* и *w* копируются массивы *z_res.x* и *w_res.x* соответственно (где переменная *x* есть принадлежащий классу *scipy.optimize.OptimizeResult* массив значений типа *float*, представляющий решение задачи линейного программирования); в противном случае выводится сообщение об отсутствии решения.

Затем посредством переменной *v* типа *float* вычисляется значение матричной игры посредством возведения в степень *-1* суммы значений массива *w*, в переменные *p* и *q* копируются массивы *z* и *w* соответственно, предварительно умноженные на *v* функцией *arr_mult()*. После этого создаются три переменные, которые будут возвращаться функцией: *p_res*, *w_res*, *v_res*. Массивы *p_res* и *w_res* (оптимальные стратегии в виде дробей) есть результаты работы функции *frac(p)*, *frac(q)* с элементами типа *Fraction*, представляющими пары чисел: числитель и знаменатель дроби, ранее записанной в *float*. Переменная *v_res* (решение матричной игры в виде дроби) тоже типа *Fraction*, получаемая приведением *v* сперва к типу *tuple* функцией *as_integer_ratio()*, а затем к *Fraction*, знаменатель которого ограничивается сверху числом 1000000 функцией *limit_denominator()* по умолчанию.

В итоге результаты возвращаются в массиве в следующем порядке: *v_res*, *p_res*, *q_res*.

matrix_negative(a)

Библиотек не использует.

Ввод. Матрица *a*.

Задача. Создать матрицу на основе вводной, только с элементами, умноженными на *-1*.

Возвращаемое значение. Возвращает новосозданную матрицу.

Работа функции. Функция использует два локальных массива для создания: *temp* и *res*. Первый массив используется для временного накопления строки с элементами соответствующей строки, умноженными на *-1*, которая потом добавляется в результирующий массив, из которого составляется матрица на возвращение.

matrix_t(a)

Библиотек не использует.

Ввод. Матрица *a*.

Задача. Создать матрицу, являющуюся транспонированной вводной матрицей.

Возвращаемое значение. Транспонированная матрица.

Работа функции. Алгоритм работы функции аналогичен алгоритму работы предыдущей функции *matrix_negative()* за исключением того, что в *temp* набираются не строки вводной матрицы, а столбцы, при том элементы не умножаются при добавлении на *-1*.

frac(a)

Используемые библиотеки. *fractions* (импортируется один класс *Fraction*).

Ввод. Массив *a*, состоящий из элементов типа *float*.

Задача. Создаёт массив из элементов типа *Fraction*, представляющий числа с плавающей точкой из вводного массива в виде дробей.

Возвращаемое значение. Возвращает массив из элементов типа *Fraction*.

Работа функции. Алгоритм аналогичен методу вычисления переменной *v_res* из функции *nash_equilibrium(a)* за исключением того, что результат преобразования *float* элемента к типу *Fraction* сразу записывается в массив.

form_ans(cost, p_in, q_in)

Используемые библиотеки. *matplotlib.pyplot* (в сокращённом виде *plt*).

Ввод. *cost* — объект типа *Fraction*, является представлением решения матричной антагонистической игры в виде дроби; *p_in* — массив элементов типа *Fraction*, является представлением оптимальной стратегии первого игрока в матричной антагонистической игре в виде дробей; *q_in* — массив элементов типа *Fraction*, является представлением оптимальной стратегии второго игрока в матричной антагонистической игре в виде дробей.

Задача. Показать решение матричной антагонистической игры и оптимальные смешанные стратегии обоих игроков в виде дробей; визуализировать спектры оптимальных смешанных стратегий обоих игроков.

Возвращаемого значения нет.

Работа функции. Сперва выводятся значение игры, оптимальная стратегия сперва первого, а затем и второго игрока. Затем с помощью *fig = plt.figure(figsize = (10, 10))* создаётся пространство размером 10 на 10 дюймов (объект типа *Figure*) для размещения графиков. После этого с помощью *fig.add_subplot()* два графика *ax1* и *ax2* (для каждого из них данная функция вызывалась с параметрами *211* и *212* соответственно, где *21** значит, что пространство делится пополам на два по горизонтали, а ***1* и ***2* определяют порядок расположения графиков, и таким образом первый график будет расположен над вторым), и для того, чтобы изобразить спектры оптимальных стратегий на этих графиках, посредством функции *ax*.stem()* на графике *ax** (** = 1* для *p*, ** = 2* для *q*) выставляются точки оптимальных стратегий первого и второго игроков, где порядковый номер значений соответствует горизонтальной оси, а само значение — вертикальной.

prac2test()

Используемые библиотеки. *unittest*, *prac2* (файл, содержащий основную программу), *fractions* (импортируется один класс *Fraction*)

Ввод отсутствует.

Задача. Проверить работу функции *nash_equilibrium()* на корректность четырьмя тестами.

Возвращаемого значения нет.

Работа программы. Осуществляется через использование класса на основе библиотеки *unittest*, где в каждом из четырёх тестов для тестирования используется функция *assertequal()*, где сперва записана проверяемая функция с проверяемым вводом, а затем ожидаемый от работы функции ответ. Ответы записаны в типе *Fraction*, поскольку функция возвращает объекты этого же типа.